INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Sensibilité de mesures transitoires des réseaux d'automates stochastiques : approche parallèle

Haïscam ABDALLAH, Moulaye HAMZA

N°4190

Juin 2001

THÈMES 1 et 4

ISSN 0249-6399 ISRN INRIA/RR--4190--FR

Sensibilité de mesures transitoires des réseaux d'automates stochastiques : approche parallèle

Haïscam ABDALLAH*, Moulaye HAMZA†

Thèmes 1 et 4 — Réseaux et systèmes — Simulation et optimisation de systèmes complexes
Projets Aladin et Armor

Rapport de recherche n°4190 — Juin 2001 — 44 pages

Résumé : Les Réseaux d'Automates Stochastiques (RAS) constituent un outil de prédilection pour modéliser le comportement des systèmes et réseaux informatiques, en particulier les systèmes parallèles. En vue de mener une étude transitoire permettant d'obtenir les mesures de performance et la sensibilité de certaines d'entre elles, nous sommes confrontés aux problèmes de complexités temporelle et spatiale (espace mémoire) ainsi qu'au contrôle de la précision des résultats numériques obtenus. L'intérêt des RAS est qu'ils permettent d'éviter la construction du générateur infinitésimal et un affrontement de la complexité temporelle grâce aux propriétés de l'algèbre tensorielle.

L'objectif de cette étude est d'abord le calcul de la sensibilité du vecteur des probabilités d'état des RAS en régime transitoire. Une application de cette démarche à l'espérance de la récompense cumulée sur un intervalle aura lieu. Nous avons sélectionné et modifié la méthode (stable) de l'uniformisation pour calculer cette sensibilité de façon séquentielle. Ensuite, nous avons étudié l'apport du parallélisme face à l'accroissement du temps de calcul avec la raideur en développant un algorithme parallèle relatif à cette méthode.

* Projet ALADIN
† Projet ARMOR

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Cet algorithme s'avère efficace et permet de traiter, en un temps de calcul acceptable, des systèmes avec plus d'un million d'états et une grande valeur du temps de mission.

Mots-clé : Systèmes parallèles, réseaux d'automates stochastiques, solution transitoire, sensibilité, uniformisation, parallélisme.

(Abstract: pto)

Sensitivity analysis of the transient measures of stochastic automata networks : parallel approach

Abstract: Analysis of stochastic automata networks (SAN) is a well established approach for modeling the behaviour of computing networks and systems, particularly parallel systems. The transient study of performance measures and their sensitivity leads us to time and space complexity problems as well as error control of the numerical results. The SAN theory presents some advantages such as avoiding to build the entire infinitesimal generator and facing the time complexity problem thanks to the tensor algebra properties.

The aim of this study is first the computation of the sensitivity of the transient probability state vector of SAN models. Next, an application of this analysis is done to the expected accumulated reward over an interval. we have selected and modified the (stable) uniformization method in order to compute that sensitivity in a sequential way. Then, after studying the contribution of parallelism in front of the increasing execution time for stiff models, we have proposed a parallel algorithm of that technique. The latter algorithm proves to be efficient and allows to process, within a fair computing time, systems with more than one million states and a large mission time value.

Key-words: Parallel systems, stochastic automata networks, transient solution, sensitivity, uniformization, parallelism.

1 Introduction

Les modèles markoviens constituent une bonne approche pour analyser la performance, la fiabilité et la performabilité de systèmes informatiques. Le comportement de ces systèmes est modélisé en utilisant des techniques de spécification comme les réseaux de files d'attente (QN pour Queueing Networks), les réseaux de Petri stochastiques généralisés (GSPN pour Generalized Stochastic Petri Nets) et bien d'autres encore [1]. Malheureusement, la taille des Chaînes de Markov à Temps Continu (CMTC) homogène résultant de la modélisation de nombreux systèmes réels peut être extrêmement importante, atteignant dans certains cas des millions d'états. Ainsi, malgré une bonne connaissance de la théorie des CMTC et l'existence de nombreux algorithmes dans ce domaine, les limites des ordinateurs peuvent être très vite atteintes lorsqu'il s'agit de traiter ce genre de modèles. Un moyen de prendre en compte la taille de l'espace des états est d'exploiter la structure du système dans la description du modèle. C'est dans cet ordre d'idées que Brigitte Plateau a introduit les Réseaux d'Automates Stochastiques (RAS) [2]. En effet, dans les systèmes que l'on modélise, plusieurs composantes évoluent souvent en parallèle (les réseaux de filles d'attente par exemple) et de façon indépendante sauf en certains points, dits de *synchronisation*. En ces points, les composantes inter-agissent entre elles. Le comportement d'une composante du système est représenté par un ensemble d'états qui constitue un *automate* (une CMTC avec un espace d'état raisonnable). Une action de la composante est représentée par une transition d'un état vers un autre. Une étiquette sur la transition permet de spécifier un taux de franchissement et une probabilité d'occurrence de l'action. Ces actions sont décrites par des matrices locales à chaque automate. Sous des hypothèses probabilistes adéquates, le comportement de l'ensemble des automates ainsi constitués peut alors être représenté par une CMTC multidimensionnelle dont les états sont ceux de l'espace produit. Il a été démontré [3] que le générateur infinitésimal de cette dernière CMTC, encore appelé *descripteur*, peut être obtenu automatiquement sous une forme compacte, en appliquant les opérateurs de l'algèbre tensorielle (ou de Kronecker) sur les matrices locales de chaque automate. Cette structure du générateur permet d'avoir un gain considérable d'espace mémoire puisqu'elle évite le stockage du descripteur tout entier. Jusqu'à présent, l'étude des RAS a été faite en régime stationnaire [4, 5, 6]. Dans ce rapport, nous nous intéressons à l'évaluation numérique de la sensibilité de mesures transitoires des RAS. Il s'agit de

traiter d'abord la sensibilité du vecteur des probabilités d'état puis de celle de l'espérance de la récompense cumulée (ERC) sur un intervalle donné. Notre intérêt pour la sensibilité vient du fait que les paramètres intervenant dans la modélisation markovienne sont, la plupart du temps, estimés. Il en résulte que les mesures qui en découlent telles que la fiabilité, peuvent être sujettes à une certaine variation. Il devient alors nécessaire de déterminer la sensibilité de ces mesures aux variations des paramètres. Une telle étude peut permettre d'accorder une attention particulière à certains paramètres (les plus sensibles) ou d'en éliminer (ceux qui n'ont pas une influence significative sur les mesures) réduisant ainsi leur nombre dans le modèle.
Les problèmes auxquels nous sommes confrontés dans l'évaluation numérique de mesures transitoires et de leur sensibilité sont le temps de calcul des ordinateurs et la précision des résultats, i.e., le contrôle de l'erreur. La complexité devient très importante lorsque l'on a affaire à des CMTC raides correspondant aux systèmes hautement fiables.
Hormis l'intérêt des RAS du point de vue gain en mémoire, cette approche permet aussi d'affronter la complexité temporelle des algorithmes en exploitant les propriétés de l'algèbre tensorielle, notamment le produit vecteur matrice, où la matrice est une somme ou un produit tensoriel de plusieurs matrices de petites tailles.
La construction des algorithmes se prêtant à la parallélisation s'avère importante en vue d'affronter la croissance du temps de calcul avec la raideur.
Le contrôle de l'erreur est une nécessité absolue dans l'évaluation de mesures transitoires et leur sensibilité. Ce critère impose le choix des méthodes permettant un contrôle efficace de l'erreur globale.
Sous ces angles, nous avons évalué numériquement dans des études précédentes, la sensibilité de mesures transitoires de CMTC à espace d'état raisonnable, de taille maximale égale à 400 états [7, 8]. Des méthodes ODE L-stables ont été testées et ont montré leur efficacité vis-à-vis de la raideur. C'est aussi le cas de la méthode des puissances uniformisées (UP) dont l'avantage, par rapport aux méthodes ODE, est le contrôle de l'erreur globale de précision.
La méthode de l'uniformisation standard (SU) a montré son intérêt pour les modèles non raides même quand la taille est importante. Cette efficacité s'est trouvée réduite lorsque la raideur augmente. Le principal avantage de cette technique sur les méthodes ODE est la majoration de l'erreur globale et la détermination de la complexité *a priori*. L'étude effectuée a permis de résoudre (séquentiellement) le problème de calcul de la sensibilité dans le cas très particulier où les automates sont

indépendants, même si la raideur est extrême et le temps de mission du système est long (voir section 3).

Dans cet analyse, nous nous proposons d'adapter la méthode SU aux RAS. Il s'agit de construire dans un premier temps des algorithmes permettant un calcul séquentiel de la sensibilité de mesures transitoires pour les modèles non raides.
Dans un deuxième temps, nous examinons l'apport du parallélisme dans l'affrontement de la raideur en nous basant sur des algorithmes efficaces relatifs à un produit vecteur-matrice où la matrice est un produit ou une somme tensoriel de plusieurs matrices de petites tailles. Ce rapport est organisé comme suit. Après avoir rappelé quelques notions de base de l'algèbre tensorielle et le formalisme des RAS dans la section suivante, nous adaptons la méthode SU au calcul de la sensibilité du vecteur des probabilités d'état dans la section 3. La calcul séquentiel de cette sensibilité est abordé dans la section 4. Une application de l'étude est faite à la sensibilité de l'ERC dans la section 5. La section 6 est consacrée à l'implémentation parallèle de la méthode SU. Les résultats numériques obtenus sur un réseau ATM sont exposés dans la section 7. La section 8 conclut le rapport.

2 Formalisme des RAS

Dans cette section, nous rappelons d'abord les propriétés de base de l'algèbre tensorielle. Ensuite, nous montrons comment un système est décrit par un RAS et illustrons ces propos par un exemple simple.

2.1 Rappel de l'algèbre de Kronecker [9, 10]

2.1.1 Produit tensoriel

Etant données deux matrices $A \in \mathbb{R}^{n_1 \times m_1}$ et $B \in \mathbb{R}^{n_2 \times m_2}$, le produit tensoriel de A et B, noté $C = A \otimes B$, est défini par :

$$C = (a_{ij}B) \in \mathbb{R}^{n_1 n_2 \times m_1 m_2}$$

.

Exemple. Prenons deux matrices A et B telles que

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \; et \; B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

Le produit tensoriel $C = A \otimes B$ est égal à :

$$C = \begin{pmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{pmatrix}$$

$$C = \left(\begin{array}{ccc|ccc} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} & a_{12}b_{11} & a_{12}b_{12} & a_{12}b_{13} \\ a_{11}b_{21} & a_{11}b_{22} & a_{11}b_{23} & a_{12}b_{21} & a_{12}b_{22} & a_{12}b_{23} \\ a_{11}b_{31} & a_{11}b_{32} & a_{11}b_{33} & a_{12}b_{31} & a_{12}b_{32} & a_{12}b_{33} \\ \hline a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} & a_{22}b_{11} & a_{22}b_{12} & a_{22}b_{13} \\ a_{21}b_{21} & a_{21}b_{22} & a_{21}b_{23} & a_{22}b_{21} & a_{22}b_{22} & a_{22}b_{23} \\ a_{21}b_{31} & a_{21}b_{32} & a_{21}b_{33} & a_{22}b_{31} & a_{22}b_{32} & a_{22}b_{33} \end{array}\right)$$

2.1.2 Somme tensorielle

La somme tensorielle des deux matrices **carrées** $A \in \mathbb{R}^{n_1 \times n_1}$ et $B \in \mathbb{R}^{n_2 \times n_2}$, notée $A \oplus B \in \mathbb{R}^{n_1 n_2 \times n_1 n_2}$, est une somme de deux produits tensoriels:

$$A \oplus B = A \otimes I_{n_2} + I_{n_1} \otimes B$$

Exemple. La somme tensorielle des deux matrices A et B précédentes est égale à:

$$A \oplus B = \left(\begin{array}{ccc|ccc} a_{11}+b_{11} & b_{12} & b_{13} & a_{12} & 0 & 0 \\ b_{21} & a_{11}+b_{22} & b_{23} & 0 & a_{12} & 0 \\ b_{31} & b_{32} & a_{11}+b_{33} & 0 & 0 & a_{12} \\ \hline a_{21} & 0 & 0 & a_{22}+b_{11} & b_{12} & b_{13} \\ 0 & a_{21} & 0 & b_{21} & a_{22}+b_{22} & b_{23} \\ 0 & 0 & a_{21} & b_{31} & b_{32} & a_{22}+b_{33} \end{array}\right)$$

2.1.3 Propriétés de base

- Associativité

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C$$
$$A \oplus (B \oplus C) = (A \oplus B) \oplus C$$

- Distributivité par rapport à la somme ordinaire

$$(A + B) \otimes (C + D) = (A \otimes C) + (B \otimes C) + (A \otimes D) + (B \otimes D)$$

- Compatibilité avec la multiplication

$$(A \times B) \otimes (C \times D) = (A \otimes C) \times (B \otimes D)$$

- Compatibilité avec transposition des matrices

$$(A \otimes B)^T = A^T \otimes B^T$$

- Compatibilité avec l'inversion des matrices
  Si A et B sont deux matrices carrées inversibles,

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$$

Notons que $\otimes$ et $\oplus$ ne sont pas commutatifs.
La propriété d'associativité pour les produits et les sommes tensoriels permet d'affirmer que les opérateurs

$$\bigotimes_{i=1}^{N} B^{(i)} \; et \; \bigoplus_{i=1}^{N} B^{(i)}$$

sont bien définis. En particulier, la somme tensorielle de N matrices peut s'écrire comme la somme ordinaire de N termes, chacun de ces derniers étant un produit tensorielle de N matrices

$$\bigoplus_{i=1}^{N} B^{(i)} = \sum_{i=1}^{N} I_{n_1} \otimes \ldots \otimes I_{n_{i-1}} \otimes B^{(i)} \otimes I_{n_{i+1}} \otimes \ldots \otimes I_{n_N}, \qquad (1)$$

où n_k est l'ordre de la matrice carrée $B^{(k)}$ et I_{n_k} est la matrice identité d'ordre n_k. Supposons par exemple que $B^{(1)} \in \mathbb{R}^{n_1 \times n_1}$ et $B^{(2)} \in \mathbb{R}^{n_2 \times n_2}$ soient deux matrices de transition (resp. générateurs infinitésimaux) de deux chaînes de de Markov *indépendantes* à temps discret (resp. continu). La chaîne de Markov définie sur *l'espace produit* admet pour matrice de transition (resp. générateur infinitésimal) le produit tensoriel (resp. somme tensorielle) de $B^{(1)} \in \mathbb{R}^{n_1 \times n_1}$ et $B^{(2)} \in \mathbb{R}^{n_2 \times n_2}$. Dans le cas où les chaînes ne sont pas indépendantes, les dépendances peuvent être exprimées au moyen de produits tensoriels de certaines matrices complémentaires (voir section suivante).

2.2 Modélisation par les RAS [6, 11]

Les RAS constituent un formalisme pour la modélisation des systèmes à grand espace d'état et la résolution de ces modèles. L'idée de base de cette approche est de décrire un système comme un ensemble de sous-systèmes. Chacun de ces sous-systèmes est vu comme un automate stochastique. La modélisation RAS s'applique aussi bien aux processus à temps discret qu'aux processus à temps continu.
Nous limiterons notre étude aux CMTC.

2.2.1 Les transitions

L'état d'un RAS est défini comme la combinaison de tous les états internes de chaque automate. Le changement de l'état global d'un RAS peut être le résultat de deux type d'événements.

- **Un événement local**. Ce genre d'événement ne concerne qu'un seul automate. Il représente le changement de l'état interne de cet automate. C'est donc une *transition locale* à l'automate considéré. L'automate est décrit par une CMTC avec son propre générateur infinitésimal local.
- **Un événement synchronisant**. L'occurrence d'un tel événement correspond au changement simultané de l'état interne d'au moins deux automates puisqu'il se produit aux points de synchronisation (ou de rendez-vous). Un événement synchronisant produit donc une *transition de synchronisation*. Cette dernière n'est possible que si et seulement si tous les automates concernés par l'événement sont au point de rendez-vous, donc prêts pour la tran-

sition. Les transitions locales peuvent être représentées simplement par leur taux de transition. Quant aux transitions de synchronisation, leur représentation nécessite des informations sur l'événement synchronisant, le taux de franchissement et la probabilité d'occurence. On utilisera donc un *triplet de synchronisation* composé de ces informations.

Notons qu'un taux de transition dans un automate peut être fonction de l'état des autres automates. On dit alors qu'il existe une dépendance fonctionnelle entre les automates. **Dans ce travail, nous ne traitons pas de la dépendance fonctionnelle**.

2.2.2 Le descripteur

Le *descripteur (markovien)* est le nom donné au générateur infinitésimal de la chaîne de Markov associé au système tout entier. Il s'exprime sous forme compacte, en fonction des générateurs infinitésimaux des automates (les transitions locales) et d'un ensemble de matrices provenant des événements synchronisants. Pour définir ces dernières matrices, on choisit, pour chaque événement, un automate *principal* (ou maître), les autres étant considérés *secondaires* ou (esclaves). Ensuite, à chaque couple (événement synchronisant, automate) on fait correspondre une paire de matrices. L'une de ces matrices, dite *positive* (car ne contenant que des termes positifs), reconstitue l'occurrence de l'événement dans l'automate considéré. L'autre, dite *négative* (par opposition à la première), sert à faire l'ajustement diagonal correspondant aux taux exprimés dans la matrice positive. Etant donné que les automates non concernés par un événement synchronisant ne changent pas d'état lors de l'occurrence de cet événement, leurs matrices positives et négatives sont réduites à la matrice identité. Les matrices positives de l'automate principal contiennent les taux d'occurrence de l'événement synchronisant. Ces taux sont multipliés par les probabilités d'occurrence, le cas échéant. Ils sont fixés à 1 pour les matrices des automates secondaires. Les matrices négatives ont des taux négatifs pour l'automate principal et des taux égaux à 1 pour les automates secondaires.

Le descripteur sera donné par la somme ordinaire de matrices représentant une partie locale et une partie synchronisante. La partie locale s'obtient par la somme tensorielle des générateurs locaux de chacun des automates. A chaque événement, on fait correspondre deux produits tensoriels : un pour les matrices positives et l'autre pour les matrices négatives. La somme de toutes ces matrices constitue alors la partie synchronisante.

Un petit exemple. La figure 1 présente un RAS composé de deux automates notés $\mathcal{A}^{(1)}$ et $\mathcal{A}^{(2)}$, ayant respectivement 3 et 2 états. Dans cet exemple, il existe un seul

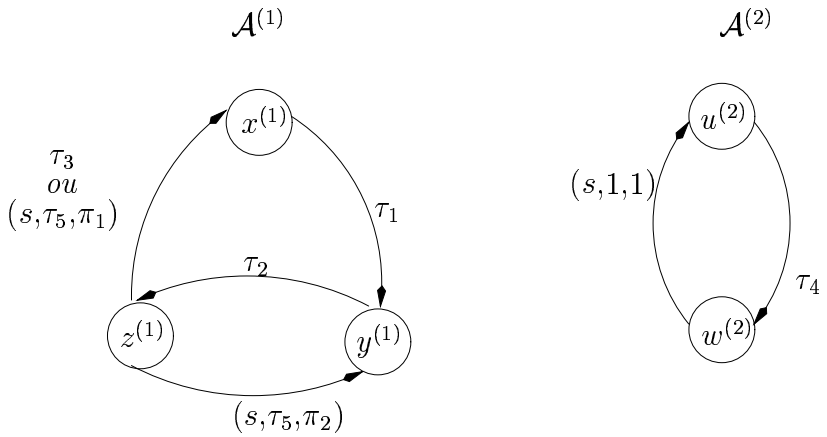

FIG. 1 – *Un exemple de RAS avec deux automates et un événement*

événement synchronisant s. Cet événement ne peut avoir lieu que si l'automate $\mathcal{A}^{(1)}$ se trouve dans l'état $z^{(1)}$ et l'automate $\mathcal{A}^{(2)}$ dans l'état $w^{(2)}$. Dans l'automate $\mathcal{A}^{(1)}$, la transition de $z^{(1)}$ vers $x^{(1)}$ peut se faire par un événement local (avec un taux τ_3) ou par l'événement synchronisant s. Lorsque cette transition a lieu grâce à s, l'état suivant est choisi avec une probabilité d'occurrence (ici π_1 ou π_2). Cela conduit aux deux possibiltés de transitions :

- soit $z^{(1)} \to x^{(1)}$ et $w^{(2)} \to u^{(2)}$ avec un taux $\tau_5\pi_1$
- soit $z^{(1)} \to y^{(1)}$ et $w^{(2)} \to u^{(2)}$ avec un taux $\tau_5\pi_2$

Le modèle global correspondant à cet exemple est donné sur la figure 2. Chaque état est représenté par un couple d'états relatifs aux automates 1 et 2 respectivement. Pour cet exemple, en choisissant l'automate 1 comme automate principal, on obtient les matrices suivantes :
Partie locale:

$$Q_l = Q_l^{(1)} \oplus Q_l^{(2)} = \begin{pmatrix} -\tau_1 & \tau_1 & 0 \\ 0 & -\tau_2 & \tau_2 \\ \tau_3 & 0 & -\tau_3 \end{pmatrix} \oplus \begin{pmatrix} -\tau_4 & \tau_4 \\ \tau_5 & -\tau_5 \end{pmatrix}$$

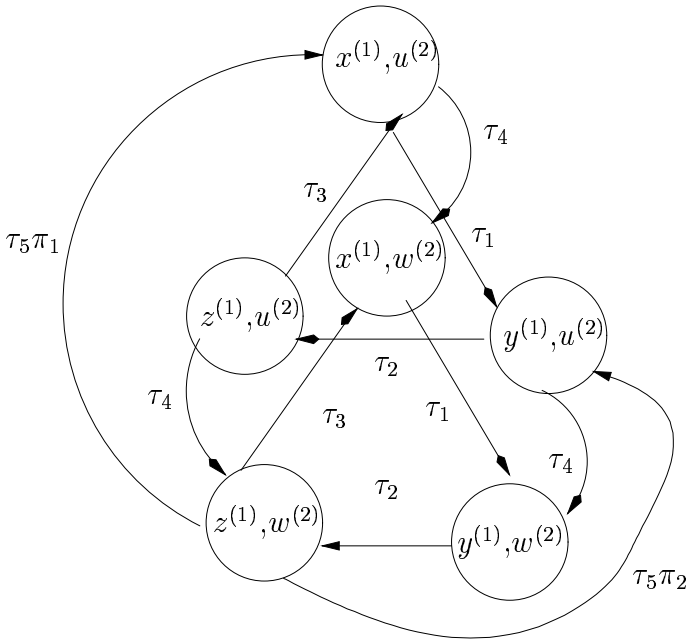

FIG. 2 – *Le modèle global*

$$Q_l = \left(\begin{array}{cc|cc|cc} -(\tau_1+\tau_4) & \tau_4 & \tau_1 & 0 & 0 & 0 \\ 0 & -\tau_1 & 0 & \tau_1 & 0 & 0 \\ \hline 0 & 0 & -(\tau_2+\tau_4) & \tau_4 & \tau_2 & 0 \\ 0 & 0 & 0 & -\tau_2 & 0 & \tau_2 \\ \hline \tau_3 & 0 & 0 & 0 & -(\tau_3+\tau_4) & \tau_4 \\ 0 & \tau_3 & 0 & 0 & 0 & -\tau_3 \end{array}\right)$$

Partie synchronisante positive:

$$Q_{s^+} = Q_{s^+}^{(1)} \otimes Q_{s^+}^{(2)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \tau_5\pi_1 & \tau_5\pi_2 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

$$Q_{s^+} = \left(\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \tau_5\pi_1 & 0 & \tau_5\pi_2 & 0 & 0 & 0 \end{array}\right)$$

Partie synchronisante négative:

$$Q_{s^-} = Q_{s^-}^{(1)} \otimes Q_{s^-}^{(2)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\tau_5 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

$$Q_{s^-} = \left(\begin{array}{cc|cc|cc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\tau_5 \end{array}\right)$$

Le descripteur est :

$$Q = Q_l + Q_{s^+} + Q_{s^-}$$

$$Q = \left(\begin{array}{cc|cc|cc} -(\tau_1+\tau_4) & \tau_4 & \tau_1 & 0 & 0 & 0 \\ 0 & -\tau_1 & 0 & \tau_1 & 0 & 0 \\ --- & --- & --- & --- & --- & --- \\ 0 & 0 & -(\tau_2+\tau_4) & \tau_4 & \tau_2 & 0 \\ 0 & 0 & 0 & -\tau_2 & 0 & \tau_2 \\ --- & --- & --- & --- & --- & --- \\ \tau_3 & 0 & 0 & 0 & -(\tau_3+\tau_4) & \tau_4 \\ \tau_5\pi_1 & \tau_3 & \tau_5\pi_2 & 0 & 0 & -(\tau_3+\tau_5) \end{array}\right)$$

Il est clair que le descripteur a comme ordre le produit des tailles des espaces d'états associés aux CMTC décrivant les automates (ici au nombre de 6). Cet ordre augmente considérablement en fonction du nombre d'automates et de leurs tailles. Il est donc nécessaire d'envisager des techniques permettant la résolution du modèle global sans passer par la construction de Q.

3 Vecteur des probabilités d'état

Considérons un réseau de N automates stochastiques $\mathcal{A}^{(k)}$, $k = 1, \ldots, N$. Soit, pour tout $k = 1, \ldots, N$, $X^{(k)}$ la CMTC (unidimensionnelle) décrivant le comportement de l'automate $\mathcal{A}^{(k)}$, n_k la taille de son espace d'état $E^{(k)} = \{1, \ldots, n_k\}$. On suppose que les taux de transition dépendent d'un paramètre θ (taux de panne ou de recouvrement, probabilité de couverture). Soit $Q^{(k)}(\theta)$ le générateur infinitésimal de $X^{(k)}$ et $\Pi^{(k)}(0)$ le vecteur stochastique de sa distribution initiale, celui-ci est supposé indépendant de θ.
Désignons par $\Pi^{(k)}(\theta,t)$ le vecteur des probabilités d'état à l'instant t de la CMTC $X^{(k)}$ et $S^{(k)}(\theta,t)$ sa sensibilité, c'est-à-dire, sa dérivée partielle par rapport à θ.
Le modèle global (le RAS) est décrit par une CMTC multidimensionnelle $X = \{X_t,\, t \geq 0\}$ avec un espace d'états $E = \prod_{k=1}^{N} E^{(k)}$ de taille $M = \prod_{k=1}^{N} n_k$, un descripteur $Q(\theta)$ et une distribution initiale $\Pi(0)$.
A un instant t, la distribution transitoire de la CMTC X est donnée par le vecteur $\Pi(\theta,t)$; $S(\theta,t)$ étant le vecteur désignant sa sensibilité. On cherche à calculer les deux vecteurs $\Pi(\theta,t)$ et $S(\theta,t)$ pour une valeur donnée de t. Il est bien connu que

$\Pi(\theta,t)$ est solution du système homogène d'équations différentielles du premier ordre de Chapman-Kolmogorov :

$$\frac{\partial}{\partial t}\Pi(\theta,t) = \Pi(\theta,t)\mathrm{Q}(\theta); \quad \Pi(\theta,0) = \Pi(0) \text{ donné} \tag{2}$$

Le vecteur $S(\theta,t)$ s'obtient soit en dérivant le système précédent et en résolvant le système obtenu (ou les deux systèmes simultanément à l'aide des méthodes ODE par exemple) soit en dérivant par rapport à θ l'expression analytique de $\Pi(\theta,t)$, solution de (2).
Nous privilégions la deuxième direction dans la mesure où les travaux précédents ont permis de dégager des méthodes numériques robustes permettant un contrôle efficace de l'erreur globale de précision après troncature d'une somme infinie. Ces méthodes permettent aussi une détermination *a priori* de la complexité temporelle relative à des problèmes de taille et de raideur importantes [7, 12, 13, 14, 15, 16].
Le calcul des vecteurs $\Pi(\theta,t)$ et $S(\theta,t)$ pourra se faire différemment selon que les automates sont indépendants ou dépendants par synchronisation. Chacun de ces deux implique une forme spécifique du descripteur $Q(\theta)$.

3.1 Automates indépendants

Dans ce cas, le descripteur $Q(\theta)$ n'est que la somme tensorielle des N générateurs infinitésimaux $Q^{(k)}(\theta)$, $k = 1, \ldots, N$, décrivant les transitions locales :

$$Q(\theta) = \bigoplus_{k=1}^{N} Q^{(k)}(\theta)$$

La probabilité que la CMTC X soit dans l'état $i = (i_1, \ldots, i_N)$ à l'instant t, où i_k est l'état de l'automate $\mathcal{A}^{(k)}$, est alors :

$$\Pi_i(\theta,t) = \Pi_{i_1}^{(1)}(\theta,t) \times \Pi_{i_2}^{(2)}(\theta,t) \times \ldots \times \Pi_{i_{N-1}}^{(N-1)}(\theta,t) \times \Pi_{i_N}^{(N)}(\theta,t)$$

où $\Pi_{i_k}^{(k)}(\theta,t)$ est la probabilité que l'automate $\mathcal{A}^{(k)}$ soit dans l'état i_k à l'instant t. Il s'en suit que :

$$\Pi(\theta,t) = \bigotimes_{k=1}^{N} \Pi^{(k)}(\theta,t) \tag{3}$$

Le vecteur $S(\theta,t)$ se calcule par récurrence comme suit :

$$\begin{aligned}\frac{\partial}{\partial\theta}\left[\Pi^{(k-1)}(\theta,t)\otimes\Pi^{(k)}(\theta,t)\right] &= \left[\frac{\partial}{\partial\theta}\Pi^{(k-1)}(\theta,t)\right]\otimes\Pi^{(k)}(\theta,t)\\ &+ \Pi^{(k-1)}(\theta,t)\otimes\left[\frac{\partial}{\partial\theta}\Pi^{(k)}(\theta,t)\right]\\ &= S^{(k-1)}(\theta,t)\otimes\Pi^{(k)}(\theta,t)\\ &+ \Pi^{(k-1)}(\theta,t)\otimes S^{(k)}(\theta,t),\ k\geq 1 \qquad (4)\end{aligned}$$

avec $S^{(0)}(\theta,0)=0$ et $\Pi^{(0)}(\theta,0)=\Pi(0)$.
Les deux relations (3) et (4) nous ramènent au calcul des vecteurs $\Pi^{(k)}(\theta,t)$ et $S^{(k)}(\theta,t),\ k=1,\ldots,N$ pour des CMTC ayant des espaces d'état de tailles raisonnables n_k. On sélectionne ainsi les méthodes SU et UP par exemple en fonction de n_k et de l'indice de raideur $q_k t$ où q_k est le taux uniformisé de la CMTC $X^{(k)}$ ($q_k > \max_{1\leq i\leq n_k}|q_{ii}^{(k)}(\theta)|$).

3.2 Automates dépendants

Soit T le nombre total d'événements synchronisants dans le système et pour tout $i=1,\ldots,T$, $E_{i+}^{(k)}(\theta)$ la matrice de l'événement i sur l'automate $\mathcal{A}^{(k)}$, $k=1,\ldots,N$; $E_{i-}^{(k)}(\theta)$ étant sa matrice de régularisation.
Le descripteur de $Q(\theta)$ s'exprime alors comme somme ordinaire d'une partie locale et d'une partie synchronisante [17] :

$$\begin{aligned}Q(\theta) &= \bigoplus_{k=1}^{N} Q^{(k)}(\theta)+\sum_{i=1}^{T}\bigotimes_{k=1}^{N}\left[E_{i+}^{(k)}(\theta)+E_{i-}^{(k)}(\theta)\right]\\ &= \bigoplus_{k=1}^{N} Q^{(k)}(\theta)+\sum_{i=1}^{2T}\bigotimes_{k=1}^{N} E_{i}^{(k)}(\theta)\ \text{ où }\ E_{i}^{(k)}(\theta)\in\left\{E_{i+}^{(k)}(\theta),E_{i-}^{(k)}(\theta)\right\} \qquad (5)\end{aligned}$$

Notons qu'en transformant $\bigoplus_{k=1}^{N} Q^{(k)}(\theta)$ en somme ordinaire de produit tensoriels (cf. relation (1)), le descripteur $Q(\theta)$ peut alors s'exprimer sous la forme suivante :

$$Q(\theta)=\sum_{i=1}^{N+2T}\bigotimes_{k=1}^{N} Q_i^{(k)}(\theta) \qquad (6)$$

où $Q_i^{(k)}(\theta)$ désigne I_{n_i} ou $Q^{(k)}(\theta)$ ou encore $E_i^{(k)}(\theta)$.
Généralement, la forme (6) est utilisée pour résoudre le modèle global (la CMTC X) afin de ramener le problème au calcul d'un produit vecteur-matrice où la matrice est un produit tensoriel de plusieurs matrices. Nous adoptons la forme (5) pour calculer les mesures transitoires instantanées et la sensibilité de certaines d'entre elles. En effet, nous allons voir, à la fin de cette section, que cette forme permet un gain important en complexité pour le calcul de la sensibilité, par rapport à celle donnée par (6).
La solution du système (2) est :

$$\Pi(\theta,t) = \Pi(0)P(\theta,t)$$

où

$$P(\theta,t) = e^{Q(\theta)t} = \sum_{n=0}^{\infty} Q^n(\theta)\frac{t^n}{n!}$$

Compte tenu du fait que $\bigoplus_{k=1}^{N} Q^{(k)}(\theta)$ et $\sum_{i=1}^{2T} \bigotimes_{k=1}^{N} E_i^{(k)}(\theta)$ (forme (5)) sont deux générateurs infinitésimaux et que les matrices $E_i^{(k)}(\theta)$ contiennent très peu de termes non nuls, une première idée attractive pourrait consister à résoudre d'abord le problème relatif à l'indépendance des automates comme dans (3) et (4) et à envisager des méthodes spécifiques pour traiter la partie synchronisante représentée par $\sum_{i=1}^{2T} \bigotimes_{k=1}^{N} E_i^{(k)}(\theta)$. Malheureusement, les deux générateurs infinitésimaux ne commutent pas et cette idée est écartée en faveur d'une solution globale relative aux deux termes de la somme (5).
Il est important de rappeler que les méthodes UP et SU permettent un contrôle de l'erreur globale de précision du calcul de $\Pi(\theta,t)$ et $S(\theta,t)$. La méthode UP est adaptée aux CMTC raides ayant un espace d'état de taille raisonnable alors que la technique SU est efficace pour résoudre les CMTC non raides à grand espace d'état.
L'idée de base porte d'abord sur l'adaptation de la méthode SU au cas des RAS puis à l'examen de l'apport du parallélisme dans l'affrontement de la croissance du temps de calcul avec la raideur.
Rappelons que, pour $Q(\theta)$ (d'ordre M) donné entièrement, l'expression de $\Pi(\theta,t)$ donnée par la méthode SU est [15, 18] :

$$\Pi(\theta,t) = \sum_{n=0}^{\infty} p(n,qt)\Pi(0)\tilde{P}^{(n)}(\theta). \tag{7}$$

où $q > max_{1\leq i\leq M} \mid q_{ii}(\theta) \mid$ et $p(n,qt) = e^{-qt}\frac{(qt)^n}{n!}$; I étant la matrice identité d'ordre M.
La somme infinie précédente peut être tronquée à un rang N_T tel que

$$1 - \sum_{n=0}^{N_T} p(n,qt) \leq \varepsilon \tag{8}$$

où ε est une tolérance fournie a priori par l'utilisateur. Cette tolérance constitue un majorant de l'erreur globale de précision du calcul de $\Pi(\theta,t)$ par SU. Notons que pour une valeur donnée de ε, N_T reste toujours supérieur à l'indice de raideur qt.
Le vecteur de la sensibilité $S(\theta,t)$ est défini par

$$S(\theta,t) = \frac{\partial}{\partial\theta}\Pi(\theta,t) = \sum_{n=0}^{\infty} p(n,qt)\frac{\partial}{\partial\theta}\left[\Pi(0)\tilde{P}^{(n)}(\theta)\right]$$

En tronquant cette somme infinie au rang N_T, l'erreur (absolue) globale commise peut être majorée par [13, 19] :

$$t\left\|\frac{\partial}{\partial\theta}Q(\theta)\right\|\left[p(N_T,qt)+\varepsilon\right].$$

Désignons par $\tilde{\Pi}^{(n)}(\theta)$, $n = 1,\ldots,N_T$, le vecteur $\Pi(0)\tilde{P}^{(n)}(\theta)$. Ces N_T vecteurs sont calculés par la relation de récurrence suivante :

$$\tilde{\Pi}^{(n)}(\theta) = \tilde{\Pi}^{(n-1)}(\theta)\tilde{P}(\theta)\ ,\ n \geq 1\ ;\ \tilde{\Pi}^{(0)}(\theta) = \Pi(0). \tag{9}$$

Les vecteurs dérivés sont tels que :

$$\frac{\partial}{\partial\theta}\tilde{\Pi}^{(n)}(\theta) = \left[\frac{\partial}{\partial\theta}\tilde{\Pi}^{(n-1)}(\theta)\right]\tilde{P}(\theta) + \tilde{\Pi}^{(n-1)}\left[\frac{\partial}{\partial\theta}\tilde{P}(\theta)\right],\ n \geq 1; \tag{10}$$

$$\frac{\partial}{\partial\theta}\tilde{\Pi}^{(0)}(\theta) = 0\ ,\ \frac{\partial}{\partial\theta}\tilde{P}(\theta) = \frac{1}{q}\left[\frac{\partial}{\partial\theta}Q(\theta)\right]$$

Conventionnellement, pour $Q(\theta)$ et $\frac{\partial}{\partial\theta}Q(\theta)$ donnés, le calcul de $S(\theta,t)$ (et $\Pi(\theta,t)$) requiert 3 produits vecteur-matrice par itération (relation (10)). En utilisant un stockage compact pour $Q(\theta)$ et $\frac{\partial}{\partial\theta}Q(\theta)$, la complexité temporelle pourra être réduite à

$O(N_T(2\eta+\eta_s))$ où η (resp. η_s est le nombre de termes non nuls dans $Q(\theta)$ (resp. $\frac{\partial}{\partial\theta}Q(\theta)$).
L'adaptation de la méthode SU aux RAS devra réduire cette complexité temporelle en évitant de construire $Q(\theta)$ et $\frac{\partial}{\partial\theta}Q(\theta)$. Pour les RAS, la relation (9) devient :

$$\begin{aligned}\tilde{\Pi}^{(n)}(\theta) &= \tilde{\Pi}^{(n-1)}(\theta)+\frac{1}{q}\left[\tilde{\Pi}^{(n-1)}\left(\oplus_{k=1}^{N}Q^{(k)}(\theta)\right)\right] \\ &+ \frac{1}{q}\sum_{i=1}^{2T}\left[\tilde{\Pi}^{(n-1)}(\theta)\otimes_{k=1}^{N}E_i^{(k)}(\theta)\right],\ n\geq 1 \qquad (11)\end{aligned}$$

avec $\tilde{\Pi}^{(0)}(\theta)=\Pi(0)=\otimes_{k=1}^{N}\Pi^{(k)}(0)$ donné.
La dérivation par rapport à θ de l'expression précédente donne :

$$\begin{aligned}\frac{\partial}{\partial\theta}\tilde{\Pi}^{(n)}(\theta) &= \frac{\partial}{\partial\theta}\tilde{\Pi}^{(n-1)}(\theta)+\frac{1}{q}\frac{\partial}{\partial\theta}\left[\tilde{\Pi}^{(n-1)}(\theta)\oplus_{k=1}^{N}Q^{(k)}(\theta)\right] \\ &+ \frac{1}{q}\sum_{i=1}^{2T}\frac{\partial}{\partial\theta}\left[\tilde{\Pi}^{(n-1)}(\theta)\otimes_{k=1}^{N}E_i^{(k)}(\theta)\right]\ ;\ \frac{\partial}{\partial\theta}\tilde{\Pi}^{(0)}(\theta)=0 \qquad (12)\end{aligned}$$

Détaillons cette relation afin d'exprimer les dérivées du membre droit de l'égalité en fonction des $Q^{(k)}(\theta)$ et $E_i^{(k)}(\theta)$ et leurs dérivées par rapport à θ. Nous avons :

$$\begin{aligned}\frac{\partial}{\partial\theta}\left[\tilde{\Pi}^{(n-1)}(\theta)\oplus_{k=1}^{N}Q^{(k)}(\theta)\right] &= \left[\frac{\partial}{\partial\theta}\tilde{\Pi}^{(n-1)}(\theta)\right]\oplus_{k=1}^{N}Q^{(k)}(\theta) \\ &+ \tilde{\Pi}^{(n-1)}(\theta)\left[\frac{\partial}{\partial\theta}\oplus_{k=1}^{N}Q^{(k)}(\theta)\right] \qquad (13)\end{aligned}$$

Pour i fixe et $k=1,\ldots,N$, on a :

$$\begin{aligned}\frac{\partial}{\partial\theta}\left[\tilde{\Pi}^{(n-1)}(\theta)\otimes_{k=1}^{N}E_i^{(k)}(\theta)\right] &= \left[\frac{\partial}{\partial\theta}\tilde{\Pi}^{(n-1)}(\theta)\right]\otimes_{k=1}^{N}E_i^{(k)}(\theta) \\ &+ \tilde{\Pi}^{(n-1)}(\theta)\left[\frac{\partial}{\partial\theta}\otimes_{k=1}^{N}E_i^{(k)}(\theta)\right] \qquad (14)\end{aligned}$$

On peut montrer que

$$\frac{\partial}{\partial\theta}\left[\otimes_{k=1}^{N}E_i^{(k)}(\theta)\right]=\sum_{l=1}^{N}W_i^l(\theta)$$

où

$$W_i^l(\theta) = E_i^{(1)}(\theta) \otimes \ldots \otimes E_i^{(l-1)}(\theta) \otimes \left[\frac{\partial}{\partial\theta} E_i^{(l)}(\theta)\right] \otimes E_i^{(l+1)}(\theta) \otimes \ldots \otimes E_i^{(N)}(\theta)$$

D'où

$$\tilde{\Pi}^{(n-1)}(\theta) \left[\frac{\partial}{\partial\theta} \otimes_{k=1}^{N} E_i^{(k)}(\theta)\right] = \sum_{l=1}^{N} \tilde{\Pi}^{(n-1)}(\theta) W_i^l(\theta) \quad (15)$$

Du point de vue de la complexité temporelle, le calcul de $S(\theta,t)$ repose essentiellement sur N_T itérations comprenant chacune (relations (11) - (15)) :

- 2 produits vecteur-somme tensorielle de matrices
- 2T(N+1) produits vecteur-produit tensoriel de matrices.

Il est important de noter que la forme (5) permet un gain en complexité par rapport à la forme (6), notamment au niveau des produits vecteur-somme tensorielle de matrices (relation(12)). En effet, en transformant $\oplus_{k=1}^{N} Q^{(k)}(\theta)$ et $\oplus_{k=1}^{N} \frac{\partial}{\partial\theta} Q^{(k)}(\theta)$ en somme de produits tensoriels, nous effectuons $2N$ produits vecteur-produit tensoriel de matrices. En développant un algorithme spécifique pour le calcul du produit vecteur-somme tensorielle de matrices de même complexité que celui d'un produit vecteur-produit tensoriel de matrices, nous avons ainsi un rapport de gain de N par itération, soit $N_T \times N$ pour l'ensemble des calculs de $S(\theta,t)$ par la méthode SU. La section suivante comprend deux algorithmes *séquentiels* de calcul d'un produit vecteur-produit tensoriel de matrices et d'un produit vecteur-somme tensorielle de matrices. L'algorithme global du calcul de $S(\theta,t)$ par la méthode SU, basé sur les deux précédents, est exposé à la fin de cette section. Une discussion sur la stratégie de sa parallélisation est abordée dans la section 6.

4 Approche séquentielle

4.1 Produit vecteur-produit tensoriel de matrices

Intéressons nous d'abord à ce produit figurant dans les relations (14) et (15). Afin de simplifier l'écriture, nous allons nous ramener au produit

$$x \bigotimes_{k=1}^{N} A^{(k)}$$

où $A^{(k)}$, $k = 1, \ldots, N$, est une matrice carrée d'ordre n_k et x un vecteur de $M = \prod_{k=1}^{N} n_k$ éléments.
Soit $M_l^u = \Pi_{k=l}^{u} n_k$ ($M = M_1^N$) et $\bar{n}_k = M/n_k$. Considérons le schéma d'indexation suivant :

$$(l_1, l_2, \ldots, l_{N-1}, l_N) = l_{[1,N]} \rightleftharpoons (\ldots((l_1)n_2 + l_2)n_3 \ldots) = \sum_{k=1}^{N} l_k M_{k+1}^N \qquad (16)$$

avec par convention $M_{N+1}^N = 1$. En posant $A = \bigotimes_{k=1}^{N} A^{(k)}$ et $i_{[1,N]}$ (resp. $j_{[1,N]}$) comme représentation de i (resp. j), le terme général de A peut ainsi s'exprimer en fonction des termes généraux des matrices $A^{(k)}$ selon la relation suivante :

$$A_{i,j} = A_{i_{[1,N]},j_{[1,N]}} = A^{(1)}_{i_1,j_1} \times A^{(2)}_{i_2,j_2} \times \ldots \times A^{(N)}_{i_N,j_N} \qquad (17)$$

Nous allons montrer les cas où cette représentation est meilleure que la représentation classique en vue d'effectuer le produit xA.
Supposons d'abord que la matrice A, carrée d'ordre M, est construite puis stockée de façon compacte avec $\eta(A)$ éléments non nuls. La complexité temporelle du calcul de xA est en $O(\eta(A))$. Cette approche classique trouve ses limites au niveau de l'espace mémoire si A est de très grande taille.
L'expression de A comme produit tensoriel des $A^{(k)}$,$k = 1, \ldots, N$, et le stockage compact des $A^{(k)}$ permet d'éviter ce problème. On note $\eta(A^{(k)})$ le nombre d'éléments non nuls de $A^{(k)}$. L'utilisation directe de la formule (17) requiert N opérations pour calculer $A_{i,j}$. Il s'ensuit que la complexité temporelle du calcul xA est en $O(N \prod_{k=1}^{N} \eta(A^{(k)})) = O(N\eta(A))$.
Contrairement à cette approche directe, de complexité élevée, les algorithmes relatifs au produit $x \bigotimes_{k=1}^{N} A^{(k)}$ exploitent la structure inhérente d'un produit tensoriel en vue de réduire considérablement la compexité temporelle du calcul xA.

L'un des premiers algorithmes relatifs au calcul de ce produit est donné dans [17]. L'algorithme 1 (que nous nommons MP pour mélange parfait) permet aussi le calcul de xA en tenant également compte du stockage compact des matrices $A^{(k)}$ et en utilisant les permutations pour pouvoir traiter les $A^{(k)}$ séquentiellement. Cet algorithme repose sur la relation :

$$\bigotimes_{k=1}^{N} A^{(k)} = \prod_{k=1}^{N} S^T_{(M_1^k, M_{k+1}^N)} \cdot (I_{\bar{n}_k} \otimes A^{(k)}) \cdot S_{(M_1^k, M_{k+1}^N)} \qquad (18)$$

Entrée : x, n_k, $A^{(k)}$
Sortie : $Y = x \bigotimes_{k=1}^{N} A^{(k)}$

1: $n_{left} \leftarrow 1$; $n_{right} \leftarrow M_N^2$
2: **pour** $k = 1, \ldots, N$ **faire**
3: $base \leftarrow 0$; $jump \leftarrow n_k.n_{right}$
4: **si** $A^{(k)} \neq I$ **alors**
5: **pour** $offset = 0, \ldots, n_{right} - 1$ **faire**
6: $index \leftarrow base + offset$
7: **pour** $h = 0, \ldots, n_k - 1$ **faire**
8: $Z_h \leftarrow x_{index}$; $index \leftarrow index + n_{right}$
9: **fin pour**
10: $Z' = Z.A^{(k)}$
11: $index \leftarrow base + offset$
12: **pour** $h = 0, \ldots, n_k - 1$ **faire**
13: $Y_{index} \leftarrow Z'_h$
14: $index \leftarrow index + n_{right}$
15: **fin pour**
16: $base \leftarrow base + jump$
17: **fin pour**
18: $x \leftarrow Y$
19: **fin si**
20: $n_{left} = n_{left}.n_k$; $n_{right} \leftarrow n_{right}/n_{k+1}$
21: **fin pour**

Alg. 1: Algorithme MP du calcul d'un produit vecteur-produit tensoriel de matrices

où $S_{(a,b)} \in \{0,1\}^{a.b \times a.b}$ est la matrice décrivant un mélange parfait ("perfect shuffle" permutation) [6, 10] :

$$(S_{(a,b)})_{i,j} = \begin{cases} 1 & si\ j = (i \mod a).b + (i\, div\, a) \\ 0 & sinon \end{cases}$$

où mod et div désignent respectivement le modulo et la division entière.
Ainsi, un produit vecteur-produit tensoriel de matrices peut être effectué en utilisant N permutations de vecteurs et N produits du type $x\left(I_{\bar{n}_k} \otimes A^{(k)}\right)$. La matrice $\left(I_{\bar{n}_k} \otimes A^{(k)}\right)$ étant une matrice diagonale par bloc et ayant sur sa diagonale $\bar{n}_k$ fois la matrice $A^{(k)}$, la complexité temporelle de la k^{ieme} multiplication est alors en $O(\bar{n}_k \eta(A^{(k)}))$. Il en résulte que la complexité de l'algorithme MP tout entier est en

$$O\left(\sum_{k=1}^{N} \bar{n}_k \,.\, \eta\left(A^{(k)}\right)\right) = O\left(M \sum_{k=1}^{N} \frac{\eta\left(A^{(k)}\right)}{n_k}\right) \tag{19}$$

sachant que la complexité temporelle des permutations (lignes 7-10 et 12-16) est négligeable devant celle des produits vecteur-matrice. Soit $\alpha_k = \frac{\eta(A^{(k)})}{n_k}$ le nombre moyen de termes non nuls par ligne ou par colonne dans la matrice $A^{(k)}$. En supposant que $\alpha_k = \alpha$ pour tout $k \in \{1, \ldots, N\}$, l'algorithme MP est alors meilleur qu'une multiplication ordinaire xA en utilisant un stockage explicite de A si et seulement si :

$$M \times N \times \alpha < M.\alpha^N \Longleftrightarrow \alpha > N^{\frac{1}{N-1}}$$

La figure (3) montre les zones où l'algorithme MP est meilleur qu'une multiplication ordinaire en fonction de N et de α. Notons que les matrices d'événements satisfont souvent aux conditions entraînant la supériorité de l'algorithme MP.

4.2 Produit vecteur-somme tensorielle de matrices

Ce produit doit être effectué deux fois par itération (relation (13). Le problème revient à construire un algorithme de calcul du produit

$$x \bigoplus_{k=1}^{N} A^{(k)}.$$

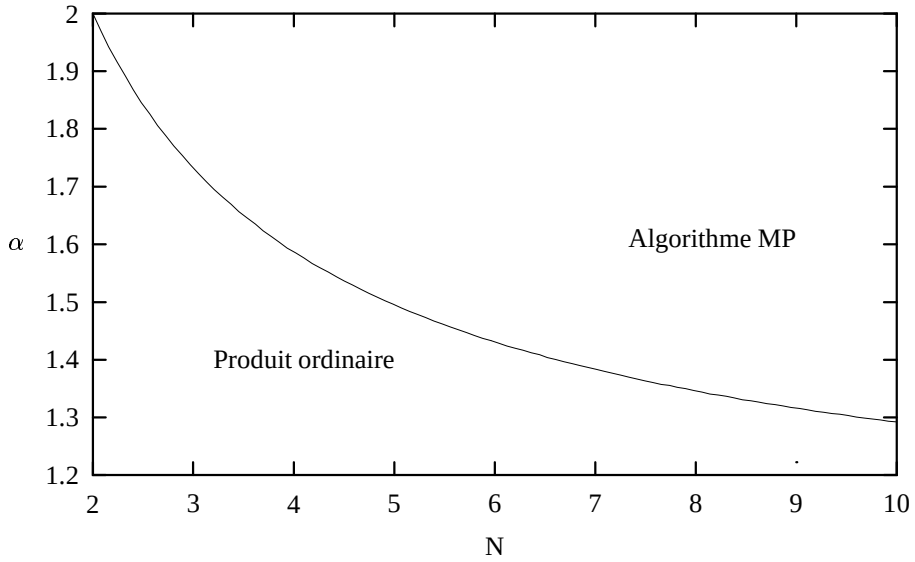

FIG. 3 – *Comparaison de la multiplication ordinaire et MP*

Rappelons que $\bigoplus_{k=1}^{N} A^{(k)}$ peut s'exprimer sous forme d'une somme de produit tensoriel (relation(1)) et que

$$\begin{aligned} x \bigoplus_{k=1}^{N} A^{(k)} &= \sum_{k=1}^{N} x \left(I_{n_1} \otimes \ldots \otimes I_{n_{k-1}} \otimes A^{(k)} \otimes I_{n_{k+1}} \otimes \ldots \otimes I_{n_N} \right) \\ &= \sum_{k=1}^{N} x \left(I_{M_1^{k-1}} \otimes A^{(k)} \otimes I_{M_{k+1}^N} \right) \end{aligned} \tag{20}$$

Pout tout $k \in \{1, \ldots, N\}$, le calcul de

$$x \left(I_{M_1^{k-1}} \otimes A^{(k)} \otimes I_{M_{k+1}^N} \right) \tag{21}$$

revient à celui de $x \bigotimes_{j=1}^{N} B^{(j)}$ avec

$$B^{(j)} = I_{n_j} \text{ si } j \neq k \text{ et } B^{(k)} = A^{(k)}$$

Entrée : n_k, $A^{(k)}$, x, n_{left}, n_{right}
Sortie : $Y = x \left(I_{M_1^{k-1}} \otimes A^{(k)} \otimes I_{M_{k+1}^K} \right)$

$base \leftarrow 0$; $jump \leftarrow n_k.n_{right}$
pour $block = 0, \ldots, n_{left} - 1$ **faire**
  pour $offset = 0, \ldots, n_{right} - 1$ **faire**
    $index \leftarrow base + offset$
    pour $h = 0, \ldots, n_k - 1$ **faire**
      $Z_h \leftarrow x_{index}$
      $index \leftarrow index + n_{right}$
    fin pour
    $Z' = Z.A^{(k)}$
    $index \leftarrow base + jump$
    pour $h = 0, \ldots, n_k - 1$ **faire**
      $Y_{index} \leftarrow Y_{index} + Z'_h$
      $index \leftarrow index + jump$
    fin pour
  fin pour
  $base \leftarrow base + jump$
fin pour

Alg. 2: Algorithme MP_k^+ pour le calcul $Y = x \left(I_{M_1^{k-1}} \otimes A^{(k)} \otimes I_{M_{k+1}^N} \right)$

L'algorithme 2, désigné par MP_k^+ (mélange parfait dans le cas d'une somme tensorielle), permet le calcul de la relation (21), en considérant $n_{left} = M_1^{k-1}$ et $n_{right} = M_{k+1}^N$. La complexité temporelle de l'algorithme MP_k^+ est en $O\left(\bar{n}_k \eta\left(A^{(k)}\right)\right)$. Il s'ensuit que la complexité temporelle du calcul de $x \bigoplus_{k=1}^N A^{(k)}$ est en

$$O\left(\sum_{k=1}^{N} \bar{n}_k \,.\, \eta\left(A^{(k)}\right)\right) = O\left(M \sum_{k=1}^{N} \frac{\eta\left(A^{(k)}\right)}{n_k}\right)$$

Cette complexité est identique à celle de $x \bigotimes_{k=1}^N A^{(k)}$ (de l'algorithme MP) donnée par la relation (19).

4.3 Algorithme global

Cet algorithme requiert d'abord le calcul du taux uniformisé q de la CMTC X tel que:

$$q > \max_{1 \le i \le M} |q_{ii}(\theta)|$$

où $q_{ii}(\theta)$ désigne le i^e élément diagonal du descripteur $Q(\theta)$. Les M éléments $q_{ii}(\theta)$ résultent d'une combinaison des éléments diagonaux des $Q^{(k)}(\theta)$ et des matrices d'événements $E^{(k)}(\theta)$, $k = 1, \ldots, N$. Pour calculer q, on part d'un vecteur initial contenant les éléments diagonaux de la première matrice $Q^{(1)}(\theta)$ et on calcule par récurrence les éléments $q_{ii}(\theta)$ de la diagonale finale. Afin de ne pas encombrer la mémoire, le vecteur utilisé servira au stockage du vecteur de la sensibilité $S(\theta,t)$. Une fois q calculé, on calcul N_T selon la relation (8) pour un ε donné. Enfin, on effectue N_T itérations selon la relation (12). L'algorithme 3 résume les différentes étapes du calcul de $S(\theta,t)$ et $\Pi(\theta,t)$. Afin d'améliorer la lisibilité de l'algorithme, nous avons volontairement omis le paramètre θ dans le texte.
Cet algorithme présente un avantage non négligeable, celui de permettre le calcul de la sensibilité du vecteur des probabilités d'état même quand le nombre des états est important, et cela, sans stocker entièrement le générateur infinitésimal. Même si l'on constate un gain en complexité temporelle, dans certains cas par rapport à l'algorithme classique, la charge de calcul du processeur demeure assez importante, en particulier, lorsque la raideur, donc qt, augmente. On peut donc se demander si le partage de cette charge de calcul entre plusieurs processeurs, i.e., la parallélisation de l'algorithme, ne pourrait pas réduire considérablement le temps d'exécution.

Entrée : $Q^{(k)}$, $\frac{\partial}{\partial\theta}Q^{(k)}$, $E_i^{(k)}$, $\frac{\partial}{\partial\theta}E_i^{(k)}$, $\Pi(0)$, t, ε
Sortie : $\Pi(\theta,t)$, $S(\theta,t)$

Calculer q
Calculer N_T
Diviser les N matrices $Q^{(k)}$ par q
Diviser les $2T$ matrices d'événements $E_i^{(1)}$ par q
$p_0 \leftarrow 1$; $\tilde{\Pi}^{(0)} \leftarrow \Pi(0)$; $S(0) = 0$; $\Pi(t) \leftarrow \Pi^{(0)}$; $S(t) = 0$ *initialisation *\
pour $n = 1, \ldots, N_T$ **faire**
  $p_n \leftarrow \frac{qt}{n} p_{n-1}$
  $MP^+(Q^{(1)}, \ldots, Q^{(N)}, \tilde{\Pi}^{(n-1)})$ *$\tilde{\Pi}^{(n)} = \tilde{\Pi}^{(n-1)} \times [I + Q/q]$*\
  pour $i = 1, \ldots, 2T$ **faire**
    $MP(E_i^{(1)}, \ldots, E_i^{(N)}, \tilde{\Pi}^{(n-1)})$
  fin pour
  $MP^+(Q^{(1)}, \ldots, Q^{(N)}, \frac{\partial}{\partial\theta}\tilde{\Pi}^{(n-1)})$ *$\frac{\partial}{\partial\theta}\tilde{\Pi}^{(n)} = \frac{\partial}{\partial\theta}\tilde{\Pi}^{(n-1)} \times [I + Q/q] + \tilde{\Pi}^{(n-1)} \times \left[\frac{\partial}{\partial\theta}\frac{Q}{q}\right]$*\
  pour $i = 1, \ldots, 2T$ **faire**
    $MP(E_i^{(1)}, \ldots, E_i^{(N)}, \frac{\partial}{\partial\theta}\tilde{\Pi}^{(n-1)})$
  fin pour
  $MP^+(\frac{\partial}{\partial\theta}Q^{(1)}, \ldots, \frac{\partial}{\partial\theta}Q^{(N)}, \tilde{\Pi}^{(n-1)})$
  pour $l = 1, \ldots, N$; $k = 1, \ldots, 2T$ **faire**
    $MP(E_k^{(1)}, E_k^{(2)}, \ldots, E_k^{(l-1)}, \frac{\partial}{\partial\theta}E_k^{(l)}, E_k^{(l+1)}, \ldots, E_k^{(N)}, \tilde{\Pi}^{(n-1)})$
  fin pour
  $\Pi(t) \leftarrow \Pi(t) + p_n\tilde{\Pi}^{(n)}$; $S(t) \leftarrow S(t) + p_n\frac{\partial}{\partial\theta}\tilde{\Pi}^{(n)}$
fin pour

Alg. 3: Algorithme séquentiel pour le calcul de $\Pi(\theta,t)$ et $S(\theta,t)$

Pour cela, il est clair que le schéma de calcul adopté doit être optimal, c'est-à-dire, demandant un faible temps de communication entre les processeurs.

5 Espérance de la récompense cumulée [8]

A la CMTC considérée dans la section 3, associons le vecteur $R = (r_i)$ des taux de récompense; r_i dénote le taux de récompense associé à l'état i de $\mathbf{X}$. L'ERC sur l'intervalle $[0,t]$ est définie par

$$E[Y(\theta,t)] = \sum_{i=1}^{M} r_i L_i(\theta,t) = L(\theta,t)R \tag{22}$$

où

$$L(\theta,t) = \int_0^t \Pi(\theta,s)ds \tag{23}$$

est la distribution cumulative de $\mathbf{X}$ sur l'intervalle $[0,t]$. La sensibilité de $E[Y(\theta,t)]$ est sa dérivée partielle par rapport à θ. Il ressort de (22) que

$$\frac{\partial}{\partial\theta}E[Y(\theta,t)] = \frac{\partial}{\partial\theta}\left[L(\theta,t)R\right] = \left[\frac{\partial}{\partial\theta}L(\theta,t)\right]R \tag{24}$$

puisque les taux de récompense sont supposés constants. Le calcul de cette sensibilité repose alors sur celui de $\frac{\partial}{\partial\theta}L(\theta,t)$, ce vecteur est noté dans la suite $S_L(\theta,t)$. L'intégration de la relation (7) donne :

$$L(\theta,t) = \sum_{n=0}^{\infty} \tilde{\Pi}^{(n)}(\theta) \int_0^t p(n,qs)ds \tag{25}$$

où, rappelons que $\tilde{\Pi}^{(n)}(\theta) = \Pi(0)\tilde{P}^{(n)}(\theta)$. On peut monter que :

$$L(\theta,t) = t\sum_{n=0}^{\infty} p(n,qt)\frac{1}{n+1}\sum_{k=0}^{n} \tilde{\Pi}^{(k)}(\theta) \tag{26}$$

Afin d'obtenir l'expression de $S_L(\theta,t)$, sensibilité de $L(\theta,t)$ par rapport à θ, nous allons dériver l'équation (26) par rapport à θ, ce qui donne

$$S_L(\theta,t) = t \sum_{n=0}^{\infty} p(n,qt) \frac{1}{n+1} \sum_{k=0}^{n} \frac{\partial}{\partial\theta} \tilde{\Pi}^{(k)}(\theta) \tag{27}$$

Les vecteurs $\frac{\partial}{\partial\theta}\tilde{\Pi}^{(k)}(\theta)$ vérifient

$$\frac{\partial}{\partial\theta}\tilde{\Pi}^{(k)}(\theta) = \left[\frac{\partial}{\partial\theta}\tilde{\Pi}^{(k-1)}(\theta)\right] \tilde{P}(\theta) + \tilde{\Pi}^{(k-1)}(\theta) \left[\frac{\partial}{\partial\theta}\tilde{P}(\theta)\right], \; k \geq 1 \tag{28}$$

La condition initiale est $\frac{\partial}{\partial\theta}\tilde{\Pi}^{(0)}(\theta) = 0$ car $\tilde{\Pi}^{(0)}(\theta) = \Pi(0)$.
En exprimant les $\tilde{P}(\theta)$ sous la forme de somme et produit tensoriel de matrices, on obtient pour $\tilde{\Pi}^{(n)}(\theta)$ et $\frac{\partial}{\partial\theta}\tilde{\Pi}^{(n)}(\theta)$ des relations similaires à (11) et (12). De ces relations, on peut déduire que **la complexité temporelle du calcul de la sensibilité de l'ERC est identique à celle du calcul de la sensibilité du vecteur des probabilités d'état**.
Notons que dans le cas d'automates indépendants, il est difficile d'exprimer $L(\theta,t)$ en fonction des distributions cumulatives des automates. Il suffit dans ce cas de prendre

$$Q(\theta) = \bigoplus_{k=1}^{N} Q^{(k)}(\theta)$$

6 Approche parallèle

Notre objectif est la parallélisation de l'algorithme global précédent, en vue d'examiner l'apport du parallélisme dans l'affrontement de la croissance du temps de calcul avec la raideur. Compte tenu de la quantité de données (matrices et vecteurs) à traiter et la liaison entre celles-ci, il est préférable d'opter pour un programme où chaque processeur possède une partie des éléments, à laquelle il applique une suite d'instructions. Cette suite d'instructions peut être choisie identique pour tous les processeurs. Cela simplifie la mise en œuvre de l'algorithme parallèle en évitant la construction de plusieurs algorithmes, un pour chaque processeur

(mode MIMD, Multiple Instruction strams, Multiple Data) ainsi que la redondance. Nous nous plaçons donc dans le cas de la construction d'un algorithme sur machine à mémoires distribuées, en mode SIMD (Simple Instruction stream, Multiple Data). Un des problèmes principaux de l'implémentation de ce genre d'algorithme parallèle est celui du contrôle de la charge des nœuds du calculateur en terme de travail à effectuer. D'où l'importance des techniques de décomposition des données et de répartition des tâches. Celles-ci doivent être réalisées de façon à *minimiser une fonction de coût liée au temps d'exécution sur p processeurs* [20].
Une application parallèle en mode SPMD sur p processeurs nécessite donc une partition des données en p sous-ensembles définissant la tâche de chaque processeur. Chaque tâche est composée principalement de phases de calcul terminée par des phases de synchronisation.
Il s'agit pour nous d'implémenter dans un premier temps les relations (11) et (12) sur une machine utilisant p processeurs. A chaque étape, ces relations sont composées essentiellement de produits vecteur-produit tensoriel de matrices et de produits vecteur-somme tensorielle de matrices. Comme dans le cas de l'algorithme séquentiel, nous traitons d'abord le produit vecetur-produit tensoriel de matrices, ensuite nous nous intéressons au produit vecteur-somme tensorielle de matrices. Dans un deuxième temps, nous introduisons les autres calculs dans l'algorithme global.

6.1 Produit vecteur-produit tensoriel de matrices

Pour calculer $Y = x \bigotimes_{k=1}^{N} A^{(k)}$ sur p processeurs, nous allons utiliser l'algorithme proposé par les auteurs de [21, 22]. Les données sont distribuées selon le schéma suivant :

- Décomposition de p en N entiers d_1,d_2,$\ldots$,d_N tel que d_k divise n_k.
  Il est à préciser que cette décomposition n'est pas unique et que toutes les décompositions sont équivalentes du point de vue du coût. Néanmoins un bon critère de différenciation serait d'obtenir une décomposition dont la somme des termes est minimum.
- Pour chaque $k \in \{1, \ldots, N\}$, former une partition de $\{1, \ldots, n_k\} = G_k$ en d_k sous-ensembles G_{kl}, $l = 1, \ldots, d_k$, i.e.,

$$\bigcup_{l=1}^{d_k} G_{kl} = G_k$$

Cette partition doit être effectuée avec un souci d'équité entre les processeurs, de sorte à assurer un équilibre des tâches.

Selon le schéma d'indexation donnée par (16), pour tout processeur p, on a la correspondance suivante :

$$p \rightleftharpoons w_{[1,N]}$$

et ainsi l'allocation des vecteurs est faite de la façon suivante :

$$w_{[1,N]} \longleftarrow Y_{(l_1,\, l_2,\ldots,\, l_N)} \;\; avec \;\; l_k \in G_{kw_k}, k = 1,\; \ldots\,, N$$

L'algorithme proposé est récursif, le calcul se faisant en N étapes.

Il est basé sur la factorisation canonique du produit tensoriel, de sorte que à chaque étape, une matrice du produit et une seule est utilisée. Un processeur utilise les données qui lui sont attribuées pour faire ses calcul, puis envoie les résultats aux processeurs qui en auront besoin pour l'étape suivante. Dans le même temps, il reçoit des autres processeurs les données dont il aura lui-même besoin pour le calcul de l'itération suivante.

Les communications sont exprimées par à l'aide de primitives de communication simples:

send : un processeur envoie un message à un seul processeur.

receive : un processeur reçoit un message d'un seul processeur.

broadcast : un processeur envoie un message à plusieurs autres processeurs.

Ces primitives sont implémentées de manière efficace sur quasiment toutes les architectures [23].

L'algorithme décrit utilise une technique de chevauchement des communications et des calculs [24], évitant ainsi de stocker tous les messages reçus (bufferisation) avant de les traiter. Un autre avantage de cet algorithme est qu'un message, partant d'un processeur, est envoyé aux seuls processeurs concernés par l'information et non à tous les autres. Le nombre de communications est ainsi considérablement réduit.

La durée d'exécution d'un programme parallèle dépend essentiellement des communications effectuées entre les processeurs. Le temps de transmission d'un message de taille $\mathcal{M}$ octets entre deux processeurs P_1 et P_2 sur un chemin $d = dist(P_1,P_2)$ est représenté par le modèle linéaire [25] :

$$t(d,\mathcal{M}) = \tau(d,\mathcal{M}) + \mathcal{M}.t_c(d,\mathcal{M})$$

$t_c(d,\mathcal{M})$ est le temps de transmission d'un octet et $\tau(d,\mathcal{M})$ est le temps d'initialisation de la communication (ou temps de latence : *start-up time*). Ce dernier dépend de principalement de d, mais ces deux paramètres peuvent être fonction de $\mathcal{M}$ si la machine utilise différents protocoles de communications pour des messages de tailles différentes (exemple d'Intel iPSC/860)
Finalement, la durée de l'exécution sera essentiellement égale au temps d'une communication (la fonction linéaire ci-dessus) multiplié par le nombre d'étape de communications.
L'algorithme du produit vecteur-produit tensoriel de matrices, que nous appelerons *PARATENS*, effectue au maximum $\Gamma(p)$ étapes de communication, $\Gamma(p)$ étant le nombre d'étapes de communications nécessaires pour envoyer un message sur p processeurs. Ce nombre dépend de types d'architecture, par exemple $\Gamma(p) = log(p)$, pour les hypercubes.
Notons que les arguments de *PARATENS* sont les matrices formant le produit tensoriel, le vecteur et les tailles de matrices.

6.2 Produit vecteur-somme tensorielle de matrices

On sait que

$$\begin{aligned}\bigoplus_{i=1}^{N} A^{(i)} &= \sum_{i=1}^{N} \left(I_{n_1} \otimes \ldots \otimes I_{n_{i-1}} \otimes A^{(i)} \otimes I_{n_{i+1}} \otimes \ldots \otimes I_{n_N}\right) \\ &= \sum_{i=1}^{N} \left(I_{M_1^{i-1}} \otimes A^{(i)} \otimes I_{M_{i+1}^{N}}\right). \qquad (29)\end{aligned}$$

Le moyen trivial de calculer $x \bigoplus_{i=1}^{N} A^{(i)}$ est d'exécuter N fois l'algorithme PARATENS précédent, en prenant comme arguments, à chaque fois toutes les matrices sauf une, égales à la matrice identité. Mais en remarquant qu'à chaque itération k de PARATENS, on calcule l'expression

$$V\left(I_{M_1^{i-1}} \otimes A^{(k)} \otimes I_{M_{i+1}^{N}}\right)$$

où V est le vecteur obtenu à l'itération précédente, alors on peut obtenir le résulat en exécutant un algorithme dont le temps de calcul est égale à celui d'une seule

exécution de PARATENS. Pour cela, il suffit de procéder comme l'indique l'algorithme 4, que nous appelons $PARATENS^+$. Dans cet algorithme, $PARATENS_k$ désigne la procédure permettant d'effectuer la k^{ieme} itération de PARATENS.

Entrée : x, n_k, $A^{(k)}$, $k = 1, \ldots, N$
Sortie : $Y = x \bigoplus_{k=1}^{N} A^{(k)}$
  $Y = 0$
  pour $k = 1, \ldots, N$ **faire**
    $Y \leftarrow Y + PARATENS_k(n_k, A^{(k)}, x)$
  fin pour

Alg. 4: Algorithme pour le calcul parallèle d'un produit vecteur-somme tensorielle de matrices

6.3 Implémentation de l'algorithme global

L'algorithme 5 calcule les vecteurs $\Pi(\theta,t)$ et $S(\theta,t)$ sur p processeurs. Comme dans les algoritmes précédents, le paramètre θ a été omis dans le but d'alléger l'écriture. La première étape de cet algorithme est le calcul du taux uniformisé q. Ce taux peut être calculé de la façon suivante. Par définition,

$$q > max_i \mid q_{ii}(\theta) \mid, i \in \{1, \ldots, M_1^N\}$$

où les $q_{ii}(\theta)$ sont les éléments diagonaux de la matrice

$$Q(\theta) = \bigoplus_{k=1}^{N} Q^{(k)}(\theta) + \sum_{i=1}^{2T} \bigotimes_{k=1}^{N} E_i^{(k)}(\theta).$$

Le descripteur $Q(\theta)$ est un générateur infinitésimal, on a donc

$$q_{ii}(\theta) = -\sum_{l=1}^{M} q_{il}(\theta)$$

où $M = M_1^N$ est la taille de la matrice carrée $Q(\theta)$.
Finalement,

$$q > max_i \mid \sum_{l=1}^{M} q_{il}(\theta) \mid, i \in \{1, \ldots, M\}$$

Entrée : $Q^{(k)}$, $\frac{\partial}{\partial\theta}Q^{(k)}$, $E_i^{(k)}$, $\frac{\partial}{\partial\theta}E_i^{(k)}$, $\Pi(0)$, t, ε
Sortie : $\Pi(\theta,t)$, $S(\theta,t)$

Calculer q et N_T
Calculer Q/q comme en séquentiel
$e_0 \leftarrow 1$; $\tilde{\Pi}^p(0) \leftarrow \Pi_0^p$; $S^p(0) = 0$; $\Pi^p(t) \leftarrow \Pi_0^p$; $S^p(t) = 0$
pour $j = 1, \ldots, N_T$ **faire**
$e_j \leftarrow \frac{qT}{j} e_{j-1}$
$PARATENS^+(Q^{(1)}, \ldots, Q^{(N)}, \tilde{\Pi}^{(j-1)})$ * $\tilde{\Pi}^{(j)} = \tilde{\Pi}^{(j-1)} \times [I + Q/q]$ *\
pour k=1, ... ,2T **faire**
$PARATENS(E_k^{(1)}, \ldots, E_k^{(N)}, \tilde{\Pi}^{(j-1)})$
fin pour* Résultat dans $\tilde{\Pi}^{(j)p}$ pour chaque processeur p *\
$PARATENS^+(Q^{(1)}, \ldots, Q^{(N)}, \frac{\partial}{\partial\theta}\tilde{\Pi}^{(j-1)})$
pour k=1, ... ,2T **faire**
$PARATENS(E_k^{(1)}, \ldots, E_k^{(N)}, \frac{\partial}{\partial\theta}\tilde{\Pi}^{(j-1)})$
fin pour
$PARATENS^+(\frac{\partial}{\partial\theta}Q^{(1)}, \ldots, \frac{\partial}{\partial\theta}Q^{(N)}, \tilde{\Pi}^{(j-1)p})$
pour l=1, ... ,N **faire**
pour k=1, ... ,2T **faire**
$PARATENS(E_k^{(1)}, E_k^{(2)}, \ldots, E_k^{(l-1)}, \frac{\partial}{\partial\theta}E_k^{(l)}, E_k^{(l+1)}, \ldots, E_k^{(N)}, \tilde{\Pi}^{(j-1)p})$
fin pour
fin pour* Résultat dans $\frac{\partial}{\partial\theta}\tilde{\Pi}^{(j)p}$ pour chaque processeur p *\
$\Pi^p(t) \leftarrow \Pi^p(t) + p_j\tilde{\Pi}^{(j)^p}$; $S^p(t) \leftarrow S^p(t) + p_j\frac{\partial}{\partial\theta}\tilde{\Pi}^{(j)p}$
fin pour

Alg. 5: Algorithme parallèle pour le calcul de $\Pi(\theta,t)$ et $S(\theta,t)$ sur P processeurs

Pour obtenir q, il suffit donc de calculer le vecteur suivant

$$Y = Q(\theta) \times \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

en remplaçant tous éléments diagonaux de $Q(\theta)$ par zéro. Ensuite choisir q tel que

$$q = 1 + \{max \mid Y_i \mid , i = 1, \ldots , M\}$$

Le calcul de $\Pi(\theta,t)$ et $S(\theta,t)$ nécessite donc une exécution de $PARATENS^+$ et $2T$ exécutions de $PARATENS$. Du point de vue de la complexité temporelle, nous avons établi en séquentiel une égalité entre la complexité temporelle de calcul du produit vecteur-produit tensoriel de matrices et du produit vecteur-somme tensorielle de matrices (cf section 4). La complexité temporelle de l'algorithme global est minimisée du fait que les algorithmes utilisés pour calculer le produit vecteur-somme ou produit tensoriel de matrices sont optimaux en nombres de communications.

7 Applications numériques

Dans cette section, nous présentons un exemple de système modélisé par un RAS. Nous exposons également les résultats numériques obtenus de l'exécution des algorithmes précédents.

7.1 Exemple d'application : le Leaky Bucket

Dans cet exemple, nous nous intéressons au contrôle de congestion d'un réseau ATM (Asynchronous Transfer Mode).
L'ATM [26] a été conçu pour faire face au transport de nouveaux types de données (téléphonie, vidéo, etc.). C'est un compromis entre le mode de transmission par paquets et les techniques synchrones. Les flux d'entrée sont découpés en paquets de longueur fixe, les cellules. Une source connectée au réseau insère des cellules d'informations selon son rythme dans le réseau, dans les espaces laissées libres par les autres sources. Les connexions hauts débits permettent d'atteindre des débits supérieurs à $600 Mbits/s$.

L'un des problèmes primordiaux dans de tels réseaux est celui de la congestion. De plus, ce problème doit être traité avec la contrainte d'une réaction et d'une adaptation rapide aux très grands débits. Ce problème est à l'origine de la perte d'information (par saturation des buffers) et de l'accroissement du temps de transmission (par augmentation du temps de séjours dans ces mêmes buffers). Le résoudre consiste à mettre ne place des méthodes préventives ou réactives permettant de réduire la congestion. Les techniques classiques ne sont pas forcément applicable, vu les débits très élevés des réseaux ATM. D'où la nécessité de générer des mécanismes de contrôle adaptés.
Le contrôle de congestion dans un réseau ATM peut s'effectuer à différents niveaux en fonction du type d'informations et des caractéristqiues du traffic. On distingue trois niveaux :

- le niveau admission
- le niveau "burst"
- le niveau cellule.

Au niveau admission, le contrôle s'effectue en vérifiant si les ressources nécessaires à la connexion sont disponibles : c'est du contrôle d'accès. Au niveau "burst", un mécanisme de contrôle (tel que le "leaky bucket") vérifie en permamence que le flot d'entrée est dans les limites du contrat négocié : c'est du contrôle de flux. Au niveau cellule, le contrôle de flux est réalisé en utilisant le bit CLP (Cell Loss Priority) contenu dans l'entête de chaque cellule. Ce bit permet de distinguer les cellules en fonction de leur priorité et de détruire prioritairement certaine d'entre elles (les cellules prioritairement détruites sont appelées cellules de basse priorité et les autres, cellules de haute priorité).
Le Leaky Bucket [27] (LB, signifiant seau troué) est un mécanisme de contrôle d'accès au réseau ATM. Ce contrôle est effectué au moyen de jetons attribués aux cellules à leur accès au réseau. La génération des jetons s'effectue à un taux égal soit au débit crête spécifié par l'utilisateur, soit à son débit moyen. Il existe plusieurs variantes de ce mécanisme. Nous nous intéressons à celle appelée le Leaky Bucket Virtuel [28].
Dans cette variante, trois tampon mémoire B_c, B_v et B_r sont utilisés. Le premier accueil les cellulles de données de l'utilisateur, les deux sont réservés aux jetons verts et rouges respectivement. Lorsque les cellules arrivent dans le tampon d'accès B_c, si ce dernier n'est pas plein, elles y sont stockées, en attente d'être servies. Servir une cellule consiste à lui attribuer un jeton vert provenant du tampon B_v.

Ce jeton constituera son autorsiation d'accès au réseau et la cellule servie est alors considérée comme conforme. Si au contraire, le tampon B_c est plein, les cellules peuvent être perdues (rejetées) alors que les ressources disponibles dans le réseau sont suffisantes pour accepter le traffic en excès sans affecter la qualité de service des autres connexions. Pour remédier à cela, on fait appel aux jetons rouges générés dans le tampon B_r. On se fixe un certain seuil S et si le tampon B_v est vide alors qu'il y a moins de S cellules dans B_c, celles-ci doivent attendre la génération de nouveaux jetons verts pour pouvoir accéder au réseau. Par contre, s'il y a plus de S cellules dans B_c lorsque B_v est vide, alors ces cellules doivent pouvoir accéder au réseau au moyen de jetons rouges, si bien-sûr, B_r n'est pas vide. La figure 4 décrit le fonctionnement de ce mécanisme.

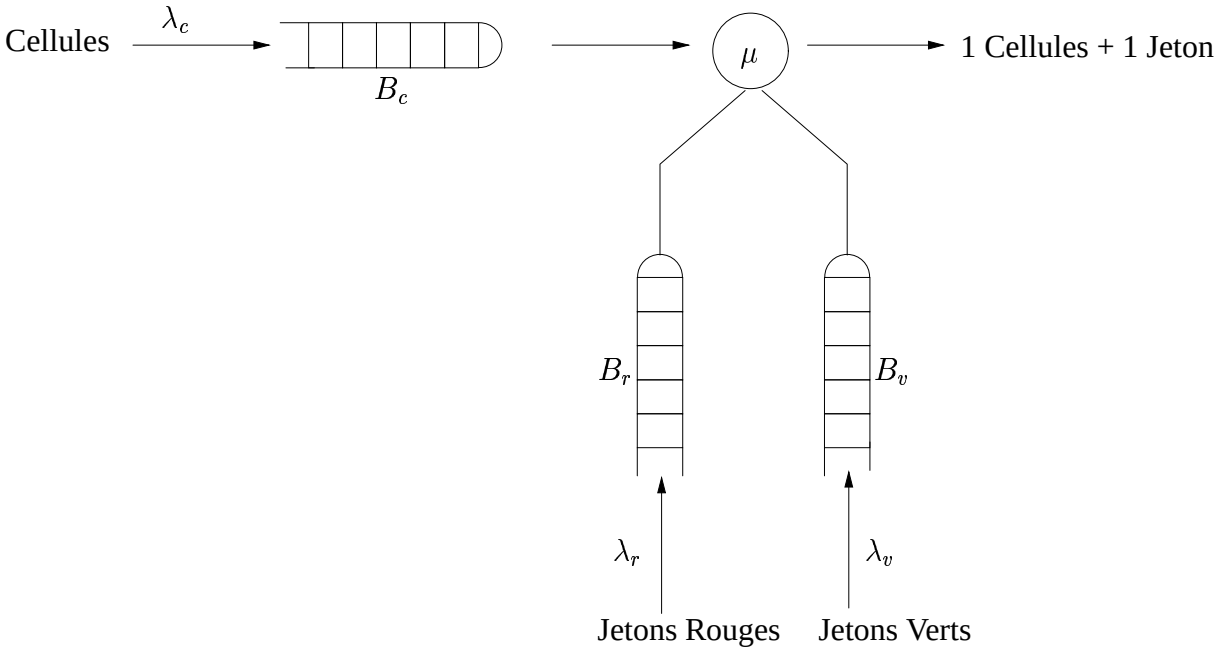

FIG. 4 – *Le leaky bucket virtuel*

Le RAS qui modélise ce mécanisme est composé de trois automates $\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)}$ et $\mathcal{A}^{(3)}$. Ces automates modéliseront respectivement le contenu des tampons B_c, B_v et B_r. On suppose que l'automate $\mathcal{A}^{(k)}$ est constitué de n_k états, $k \in \{1,2,3\}$. Ces

états sont numérotés de 0 à $n_k - 1$. Le seuil du tampon B_c est fixé à S.
Les événements se produisant dans système sont les suivants :

- Les événements locaux
  - arrivée d'une cellule avec un taux λ_c, qui est local à $\mathcal{A}^{(1)}$
  - arrivée d'un jeton vert avec un taux λ_v, qui est local à $\mathcal{A}^{(2)}$
  - arrivée d'un jeton rouge avec un taux λ_r, qui est local à $\mathcal{A}^{(3)}$
- Les événements synchronisants
  - s_1 : départ d'une cellule avec un jeton vert, avec un taux μ, agissant à la fois sur $\mathcal{A}^{(1)}$ et $\mathcal{A}^{(2)}$
  - s_2 : départ d'une cellule avec un jeton rouge, avec un taux μ, agissant à la fois sur $\mathcal{A}^{(1)}$ et $\mathcal{A}^{(3)}$

La figure 5 donne le comportement de chaque automate dans le cas où $n_1 = 4$, $n_2 = n_3 = 3$ et $S = 1$.

Les transitions dans $\mathcal{A}^{(1)}$ correspondent soit à des arrivées de cellules (donc avec un taux λ_c) soit à des départs de cellule (service). Ces départs sont des transitions de synchronisation car une cellule quitte le tampon B_c avec un jeton vert ou un jeton rouge, selon que le nombre de cellules dans est inférieur ou supérieur à S. Dans tous les cas, le taux de service est μ et la probabilité d'occurence (ou de routage) est 1, une seule alternative étant possible. Les transitions dans $\mathcal{A}^{(2)}$ et $\mathcal{A}^{(3)}$ correspondent à des arrivées (avec taux λ_v ou λ_r) de jetons verts ou rouges ou à des départs de ces jetons. Un départ de jeton signifie qu'une cellule a été servie, d'où les événements de synchronisation s_1 et s_2. Ces transitions ont pour taux μ et ont aussi une probabilité d'occurrence égale à 1. Le fait qu'un jeton rouge ne peut être utilisée que si B_v est vide est modélisé par la boucle sur l'état 0 de l'automate $\mathcal{A}^{(2)}$.
On note :

$Q^{(k)}(\theta)$ le générateur infinitésimal associé à l'automate $\mathcal{A}^{(k)}$, $i \in \{1,2,3\}$.

$E_1^{(k)}(\theta)$ la matrice d'événément positive de l'événement s_1 sur chacun l'automate $\mathcal{A}^{(k)}$, $k \in \{1,2,3\}$.

$E_2^{(k)}(\theta)$ la matrice d'événément positive de l'événement s_2 sur chacun l'automate $\mathcal{A}^{(k)}$, $k \in \{1,2,3\}$.

Le *descripteur* du système est donné par

$$Q(\theta) = Q^{(1)}(\theta) \oplus Q^{(2)}(\theta) \oplus Q^{(3)}(\theta) + E_1^{(1)}(\theta) \otimes E_1^{(2)}(\theta) \otimes E_1^{(3)}(\theta) + E_2^{(1)}(\theta) \otimes E_2^{(2)}(\theta) \otimes E_2^{(3)}(\theta)$$

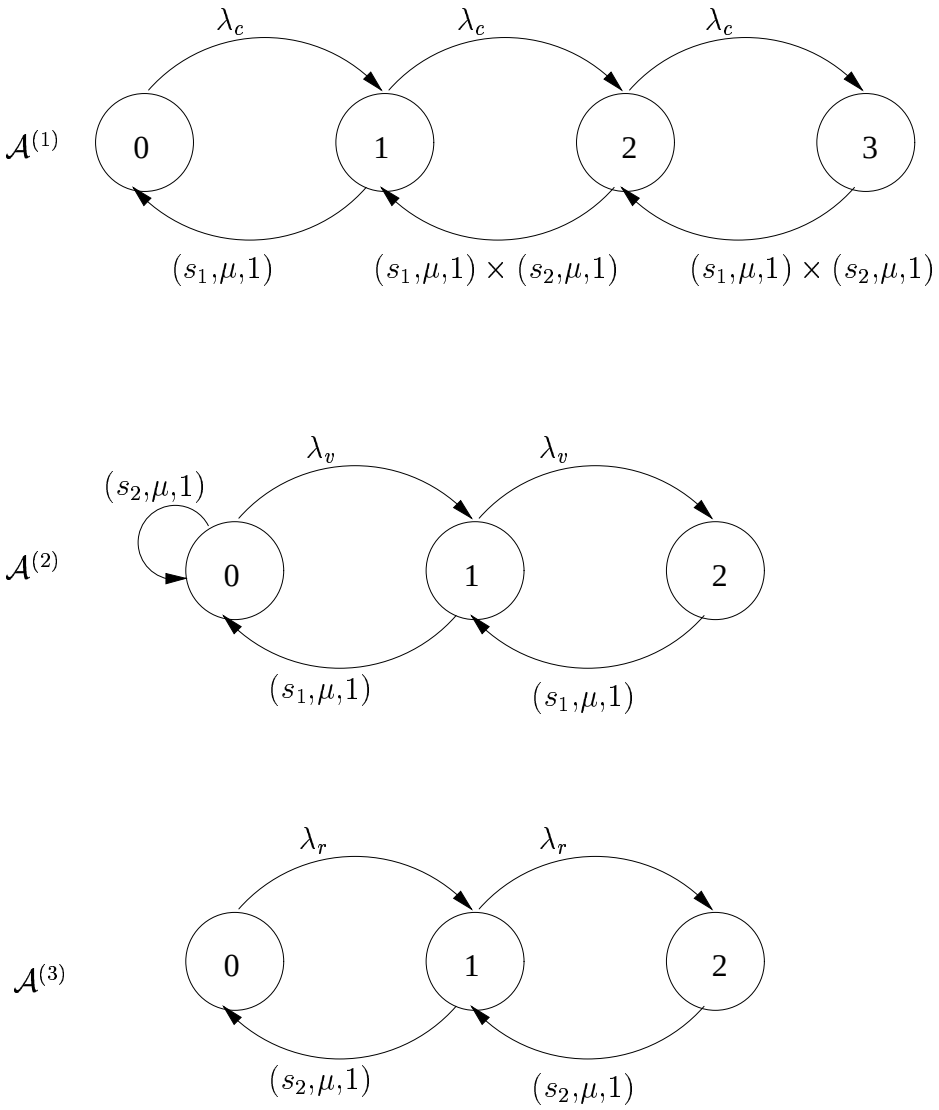

FIG. 5 – *Les diagrammes de transition des automates du RAS associé au leaky bucket virtuel*

7.2 Résultats numériques

Dans cette section, nous nous proposons d'évaluer numériquement le vecteur des probabilités d'état du système précédent et sa sensibilité par rapport à l'un des paramètres. Nous choisissons comme paramètre θ le taux de service μ. Comme le temps de calcul de l'algorithme repose essentiellement sur celui de $PARATENS$ et $PARATENS^{+}$, on peut ainsi estimer facilement le temps CPU nécessaire au calcul de l'ERC en fonction de qt et du nombre de processeurs. La taille du vecteur $S(\mu,t)$ est de $M = n_1.n_2.n_3$. Nous fixons les valeurs $n_1 = 256$, $n_2 = 128$ et $n_3 = 32$. Ce qui signifie que les tampons mémoire B_c, B_v et B_r ont des capacité limitées respectivement à 255, 127 et 31. Le seuil S est mis à 200. Le $descripteur$ ainsi obtenu est une matrice carrée d'ordre $M = 2^{20}$ et **le système a** $1.048.576$ **états**. Les taux sont tels que $\lambda_b = \lambda_c = \lambda = 0.5$ et $\mu = 1$. Le modèle réel considéré n'étant pas raide. Cependant, le raisonnement portant sur le temps de calcul de $S(\mu,t)$ ne change pas pour des rapports λ/μ très faibles.
Nous décrivons les résultats obtenus sur la machine CRAY T3E [1]. C'est une machine parallèle possédant 256 processeurs cadensés à 300 MHZ. Les calculs sont effectués en double précision.
Nous nous sommes intéressés à trois mesures pour évaluer la performance de nos algorithmes. D'abord le temps d'exécution de l'algorithme car celui-ci est très long (voir impossible à obtenir) pour certains modèles en séquentiel. Ensuite, l'accélération. Ce critère permet de mesurer l'influence du nombre de processeurs sur le temps d'exécution de l'algortithme. Si t_s est le temps obtenu en séquentiel avec le meilleur algorithme et t_p celui nécessaire à la résoluton parallèle sur p processeurs, l'accélération est donnée par le rapport :

$$\mathcal{A}_p = \frac{t_s}{t_p}$$

Enfin, nous avons calculé l'efficacité, qui met en évidence l'utilisation de chaque processeur. Cette efficacité s'obtient par :

$$\mathcal{E}_p = \frac{\mathcal{A}_p}{p} = \frac{t_s}{pt_p}$$

1. mise à notre disposition par l'Institut du Développement et des Ressources en Informatique Scientifi que (IDRIS)

Le tableau 1 nous donne les temps de calcul de $\Pi(\mu,t)$ et $S(\mu,t)$ en fonction du nombre de processeurs et de l'indice de raideur qt. La notation (e) dans ce tableau signifie que les valeurs du temps CPU correspondant à $qt = 10^5$ et $qt = 10^6$ sont estimées grâce à la connaissance a priori de la complexité temporelle de la méthode SU. Cette estimation repose sur la proportionnalité du temps CPU au nombre d'opérations. Ce tableau permet aussi de déterminer les limites de faisabilité de certains problèmes vis-à-vis du temps de calcul, en fonction de la raideur, de la taille du modèle et du nombre de processeurs.

qt	1	10	10^2	10^3	10^4	10^5 (e)	10^6 (e)
32 p	101	279	1667	12000	106507	1023317	10082377
64 p	50	173	871	6270	55649	534678	5267996
128 p	2.6	10	46	331	2933	28180	277678

TAB. 1 – *Temps d'exécution (en s) de $S(\mu,t)$ en fonction de qt et du nombre de processeurs*

Dans le tableau 2 nous avons les accélérations et l'efficacité fonction du nombre de processeurs. La valeur de l'accélération croît en fonction du nombre de processeurs ; le temps d'exécution est donc inversement proportionnel au nombre de processeurs. Par contre, l'algorithme atteint son efficacité maximale avec un nombre de processeurs compris entre 32 et 64 processeurs. Il est clair que ces valeurs sont liées à l'exemple traité et peuvent varier en fonction des cas. Cette *efficacité reste en moyenne supérieure à 0.8*, ce qui exprime une bonne utilisation des processeurs.

p	32	64	128
$\mathcal{A}_p$	28	53	101
$\mathcal{E}_p$	0.87	0.83	0.79

TAB. 2 – *Accélération et efficacité en fonction du nombre de processeurs*

8 Conclusion

Dans ce rapport, nous nous sommes intéressés au calcul de la sensibilité de mesures transitoires des RAS.
Cette approche permet de traiter le problème relatif à la taille du modèle, en évitant de construire le générateur infinitésimal global. Nous avons, dans un premier temps, adapté la méthode SU au calcul de la sensibilité du vecteur des probabilités d'état instantanées des RAS et l'avons implémenté séquentiellement. Nous l'avons ensuite appliqué au calcul de la sensibilité de l'ERC sur un intervalle donné. Dans un deuxième temps, nous avons parallélisé cette méthode en vue d'affronter la croissance du temps de calcul avec la raideur du modèle. Cette parallélisation utilise des algorithmes de produit vecteur-produit tensoriel et somme tensorielle de matrices de faibles tailles qui minimisent le temps de communication entre les processeurs. Une extension de l'étude effectuée à été faite à la sensibilité de l'ERC sur un intervalle donné. Nous avons exposé les résultats numériques portant sur un RAS modélisant un réseau ATM avec 1048000 états et une grande valeur du temps de mission. Malgré une baisse d'efficacité avec la croissane du nombre de processeurs, nous avons obtenu une valeur moyenne d'efficacité supérieure à 80%.

Références

[1] F. Neri, M. Ajmone-Marsan, and S. Donatelli. GSPN Models of Markovian Multiservere Multiqueue Systems. *Performance Evaluation*, 11:227–240, 1990.

[2] B. Plateau. On the Stochastic Structure of Parallelism and Synchronisation Model for Distributed Systems. *ACM SIGMETRICS conference On Measurement and Modeling of Computer Systems*, pages 147–153, Mai 1985.

[3] B. Plateau. De l'Evaluation du Parallélisme et de la Synchronisation. *Thèse de doctorat, Université de Paris-Sud, Orsay*, Nov 1984.

[4] P. Buchholz, G. Ciardo, S. Donnatelli, and P. Kemper. Complexity of Kronecker Operations on Sparse Matrices with Applications to Solution of Markov Models. Technical Report 97-66, ICASE, NASA Langley Research Center, Hampton, VA 23681-2199, Décembre 1997.

[5] J.M. Fourneau and al. Performance Evaluation of a Buffer Policy with Stochastic Automata Networks. *Performance Evaluation*, 1993.

[6] P. H. L. Fernandes. Méthodes Numériques pour la Solution de Systèmes Markoviens à Grands Espace d'états. *Thèse de doctorat, INPG, Grenoble*, Fev 1998.

[7] H. Abdallah et M. Hamza. Sensibilité de mesures transitoires instantanées des systèmes informatiques hautement fiables. Publication Interne 1232, IRISA, Campus de Beaulieu, Rennes, France, Février 1999.

[8] H. Abdallah et M. Hamza. Sensibilité de l'espérance de récompense cumulée des modèles markoviens raides. Rapport de Recherche INRIA RR-3904, IRISA, Campus de Beaulieu, Rennes, France, Mars 2000.

[9] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, 1994.

[10] M. Davio. Kronecker Products and Shuffle Algebra. *IEEE Trans. Comput.*, 30:116–125, 1981.

[11] B. Plateau and K. Atif. Stochastic Automata Networks for Modelling Parallel Systems. *IEEE Trans. On Software Eng.*, 17(10):1093–1108, 1991.

[12] H. Abdallah and R. Marie. The Uniformized Power Method for Transient Solutions of Markov Processes. *Computers and Operations Research*, 20(5):515–526, April 1993.

[13] H. Abdallah. Sensitivity Computation of Reliability Markov Models Using the Uniformized Power Method. *Reliability Engineering and System Safety*, 56:53–59, 1997.

[14] H. Abdallah and M. Hamza. Sensitivity analysis of instantaneous transient measures of highly reliable systems. 11^{th} *European Simulation Symposium (ESS'99), Erlangen-Nuremberg, Germany, october 26-28*, pages 652–656, 1999.

[15] H. Abdallah and M. Hamza. Sensitivity analysis of the expected accumulated reward using uniformization and IRK3 methods. *Second Conference on Numerical Analysis and Applications (NAA'2000), Rousse, Bulgaria, June 11-15*, 2000.

[16] H. Abdallah and M. Hamza. Sensitivity computation of the expected accumulated reward of stiff Markov models. *International Conference on Parallel*

and Distributed Processing Techniques and Applications (PDPTA'2000), Las Vegas, USA, 2000.

[17] B. Plateau. On the Stochastic Structure of Parallelism and Synchronisation Models for Distributed Algorithms. *Performance Evaluation*, 13(5):142–154, 1985.

[18] J. K. Muppala M. Malhotra, K. S. Trivedi. Stiffness-tolerant methods for transient analysis of stiff Markov chains. *Microelectron. Reliab.*, 34(11):1825–1841, 1994.

[19] P. Heidelberger and A. Goyal. Sensitivity Analysis of Continuous-time Markov Chains using Uniformization. *Computer Performance and Reliability*, editors: G. Iazeolla, P. J. Courtois and O. .J. Boxma, Elsevier Science Publishers B. V. Amsterdam:93–104, 1988.

[20] S. Ubéda M. Gengler and F. Desprez. *Initiation au Parallélisme*. MASSON, Paris, 1996.

[21] C. Tadonki and B. Philippe. Parallel Multiplication of a Vector by a Kronecker Product of Matrices. *Journ. of Paral. and Dist. Comput. and Pract.*, To appear.

[22] C. Tadonki and B. Philippe. Parallel Multiplication of a Vector by a Kronecker Product of Matrices (part II). *Journ. of Paral. and Dist. Comput. and Pract.*, To appear.

[23] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computing*. Prentice-Hall, Englewood Cliffs, N. J., 1988.

[24] C. Tadonki. Contribution à l'Algorithmique Parallèle. *Thèse de doctorat, Université de Rennes 1, Rennes*, Mars 2000.

[25] S.M. Müller A. Bingert, A. Formella and W. J. Paul. Isolating the Reasons for the Performance of Parallel Machines on Numerical Programs. In *International Workshop on Automatic Distributed Memory Parallelization, Automatic Data Distribution and Automatic Parallel Performace Prediction*, pages 34–64, Austin, Texas, USA, 1993.

[26] B. Weiss. *ATM*. Hermes, Paris, 1995.

[27] I. Cidon and I.S. Gopal. An approach to Integrated High-Speed Private Network. *Int. J. Digital and Analog. Caled Syst.*, 1:77–86, 1988.

[28] M. Hirano and N. Watanabe. Characteristics of a cell multiplexer for bursty atm traffic. *IEEE ICC'89*, pages 1321–1325, 1989.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399